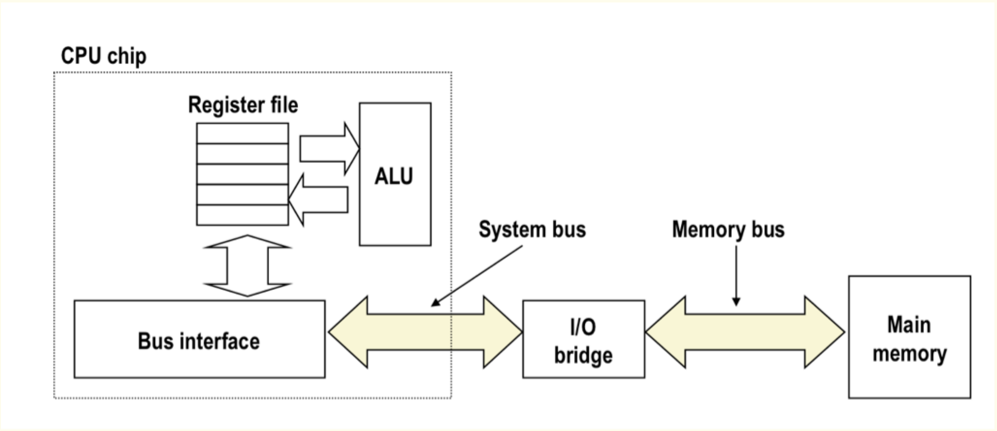
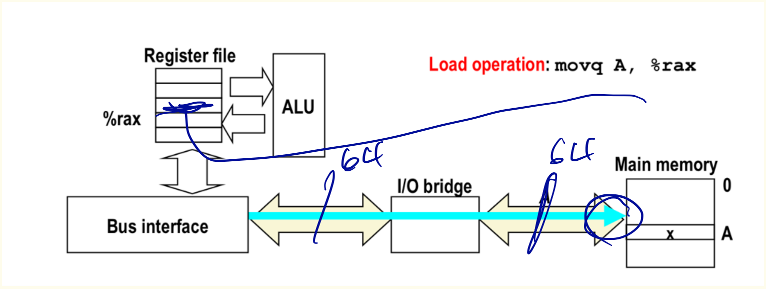
Memory Hierarchy

Where Memory really fits in

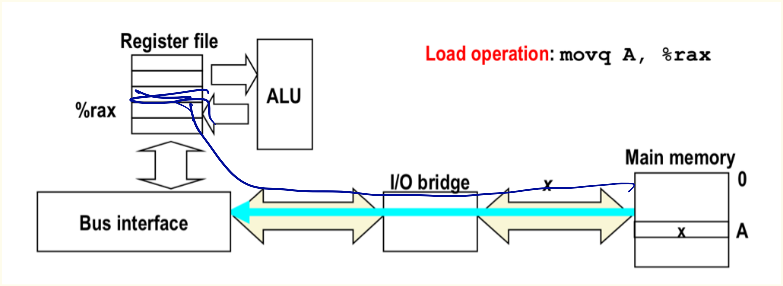


Reading from memory

①

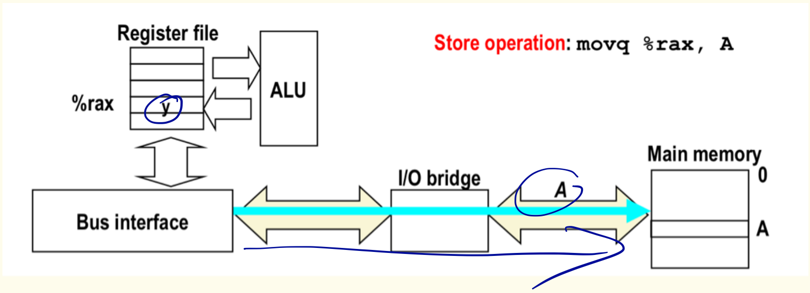


②

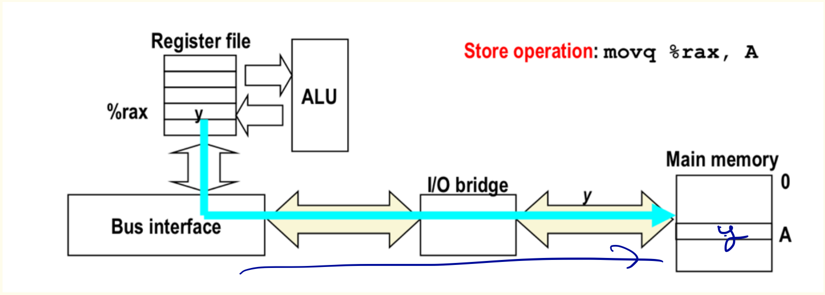


Writing to Memory

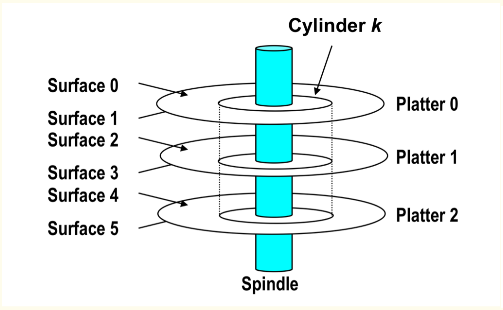
1

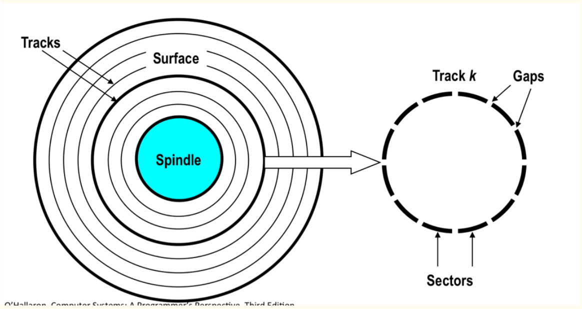


2



Disk Basics




Tracks
Surface
Spindle
Track k
Gaps
Sectors
O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

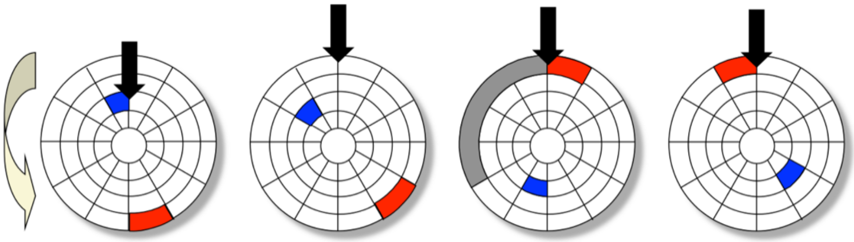
- Disks consist of platters, each with two surfaces.
- Each surface consists of concentric rings called tracks.
- Each track consists of sectors separated by gaps.

Capacity = (# bytes/sector) x (avg. # sectors/track) x (# tracks/surface) x (# surfaces/platter) x (# platters/disk)

Example:

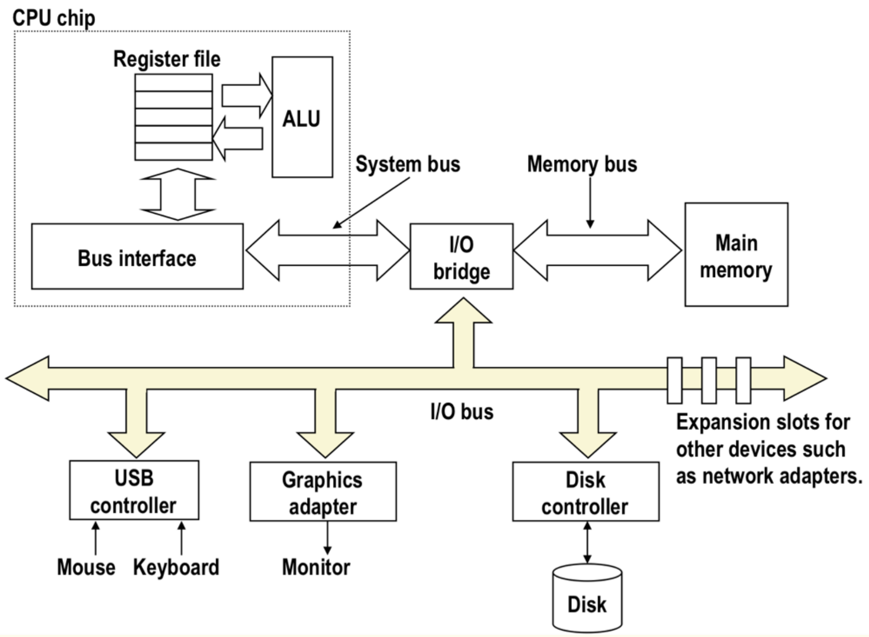
- 512 bytes/sector
- 300 sectors/track (on average)
- 20,000 tracks/surface
- 2 surfaces/platter
- 5 platters/disk

Capacity = 512 x 300 x 20000 x 2 x 5
= 30,720,000,000
= 30.72 GB

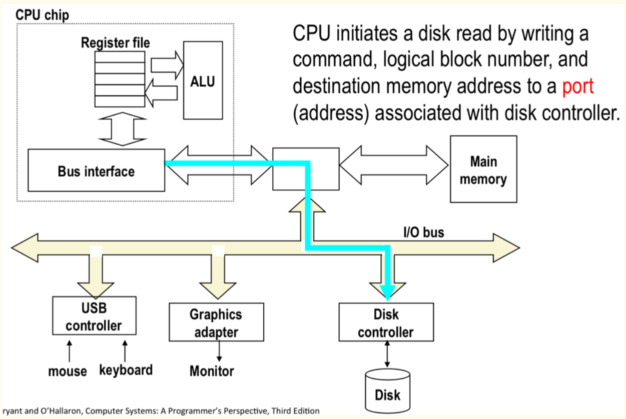


$T_{io} = T_{seek} + T_{rotate} + T_{transfer}$

- **Given:**
  - Rotational rate = 7,200 RPM
  - Average seek time = 9 ms.
  - Avg # sectors/track = 400.
- **Derived:**
  - Tavg rotation = 1/2 x (60 secs/7200 RPM) x 1000 ms/sec = 4 ms.
  - Tavg transfer = 60/7200 RPM x 1/400 secs/track x 1000 ms/sec = 0.02 ms
  - Taccess = 9 ms + 4 ms + 0.02 ms
- **Important points:**
  - Access time dominated by seek time and rotational latency.
  - First bit in a sector is the most expensive, the rest are free.
  - SRAM access time is about 4 ns/doubleword, DRAM about 60 ns
    - Disk is about 40,000 times slower than SRAM,
    - 2,500 times slower then DRAM.



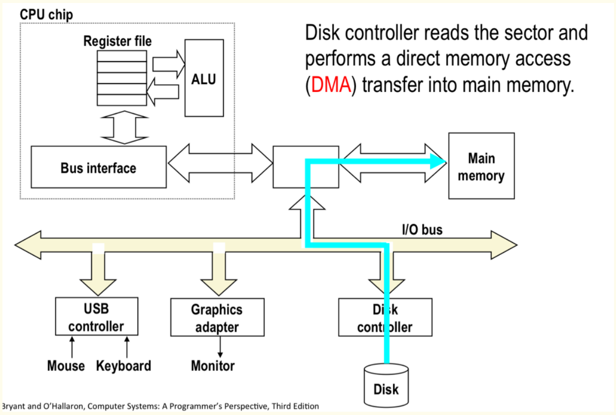
1



CPU initiates a disk read by writing a command, logical block number, and destination memory address to a port (address) associated with disk controller.

ryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

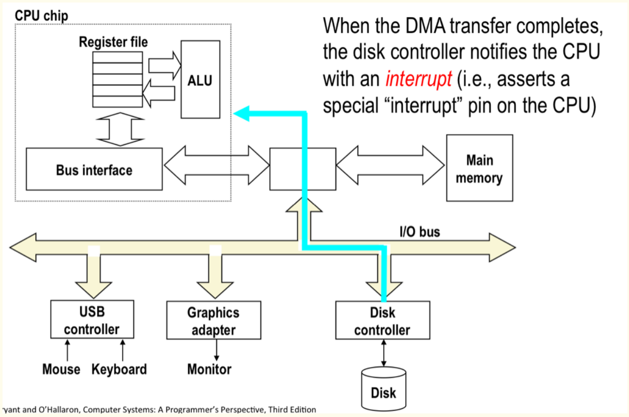
2



Disk controller reads the sector and performs a direct memory access (DMA) transfer into main memory.

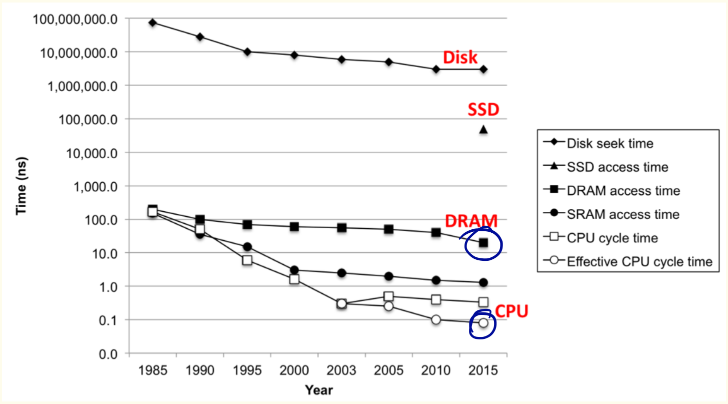
Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

3



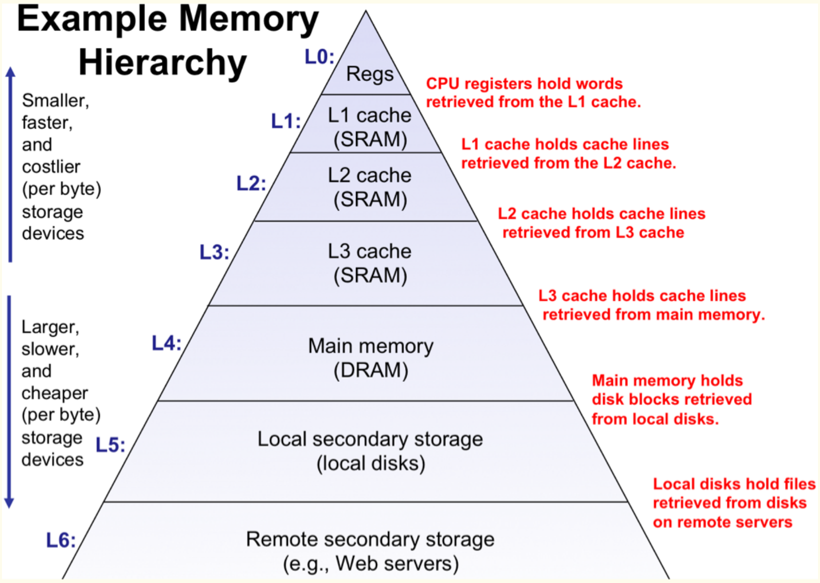
When the DMA transfer completes, the disk controller notifies the CPU with an *interrupt* (i.e., asserts a special "interrupt" pin on the CPU)

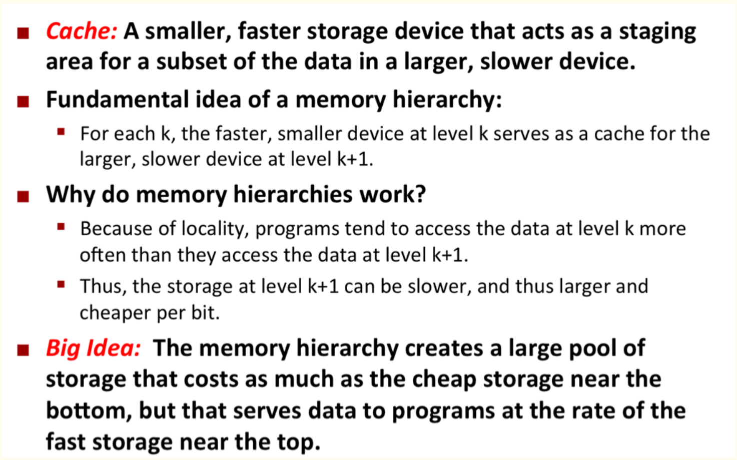
yant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition



on to locality discussion

Example Memory Hierarchy


Smaller, faster, and costlier (per byte) storage devices
Larger, slower, and cheaper (per byte) storage devices
L0:
Regs
L1:
L1 cache (SRAM)
L2:
L2 cache (SRAM)
L3:
L3 cache (SRAM)
L4:
Main memory (DRAM)
L5:
Local secondary storage (local disks)
L6:
Remote secondary storage (e.g., Web servers)
CPU registers hold words retrieved from the L1 cache.
L1 cache holds cache lines retrieved from the L2 cache.
L2 cache holds cache lines retrieved from L3 cache
L3 cache holds cache lines retrieved from main memory.
Main memory holds disk blocks retrieved from local disks.
Local disks hold files retrieved from disks on remote servers



- ***Cache:*** **A smaller, faster storage device that acts as a staging area for a subset of the data in a larger, slower device.**
- **Fundamental idea of a memory hierarchy:**
  - For each k, the faster, smaller device at level k serves as a cache for the larger, slower device at level k+1.
- **Why do memory hierarchies work?**
  - Because of locality, programs tend to access the data at level k more often than they access the data at level k+1.
  - Thus, the storage at level k+1 can be slower, and thus larger and cheaper per bit.
- ***Big Idea:*** **The memory hierarchy creates a large pool of storage that costs as much as the cheap storage near the bottom, but that serves data to programs at the rate of the fast storage near the top.**

NB: <u>Address</u> is <u>King</u>

A. Interface to memory

cpu may deal in registers, but ultimately

movq <mem>, <reg>

movq <reg>, <mem>

yields an Address to Bus

and $\text{Instruction}_{0-9} \Leftarrow \text{imem}_{10}[PC]$

Locality: Defined relative to an address

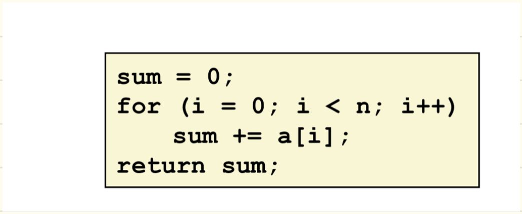
temporal

spacial

Important for all to remember b/c

$\boxed{\text{Programs with good locality run faster than programs with poor locality}}$

and it is in programmer's control to favor locality



```
sum = 0;
for (i = 0; i < n; i++)
    sum += a[i];
return sum;
```

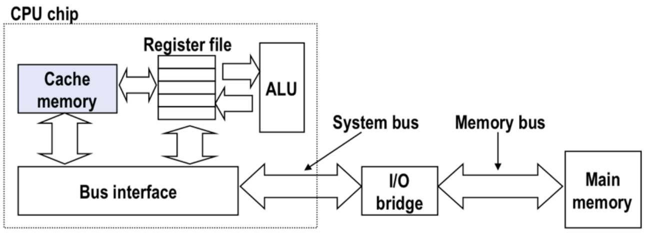
Which is better for locality, and why?

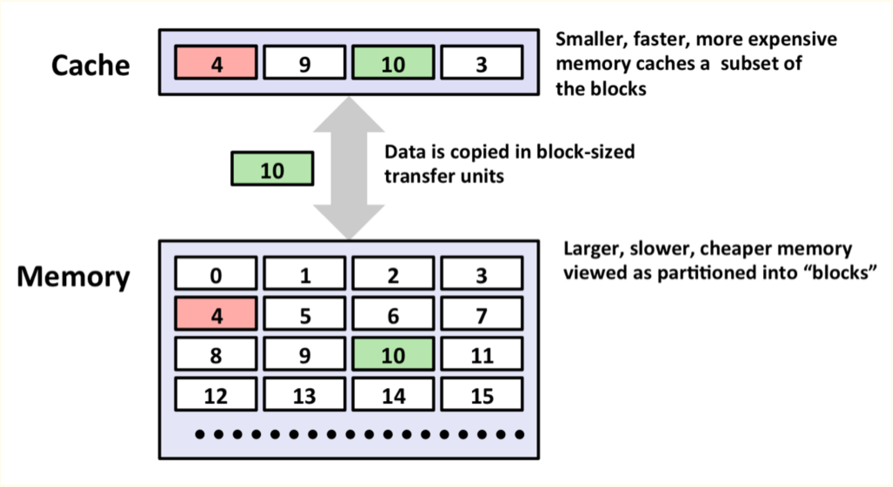
```
int sum_array_rows(int a[M][N])
{
    int i, j, sum = 0;

    for (i = 0; i < M; i++)
        for (j = 0; j < N; j++)
            sum += a[i][j];
    return sum;
}
```

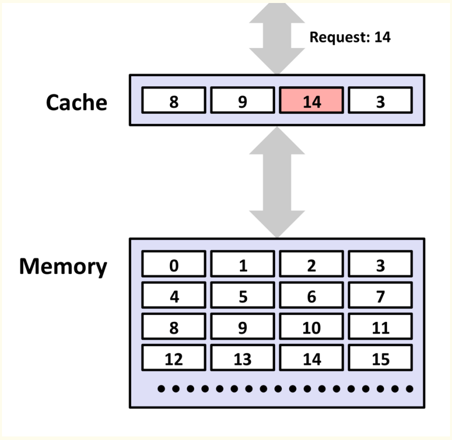
```
int sum_array_cols(int a[M][N])
{
    int i, j, sum = 0;

    for (j = 0; j < N; j++)
        for (i = 0; i < M; i++)
            sum += a[i][j];
    return sum;
}
```

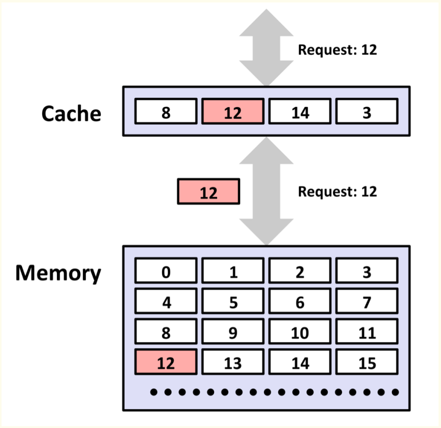
Does it really make a difference?

- **Cache memories are small, fast SRAM-based memories managed automatically in hardware**
  - Hold frequently accessed blocks of main memory
- **CPU looks first for data in cache**
- **Typical system structure:**



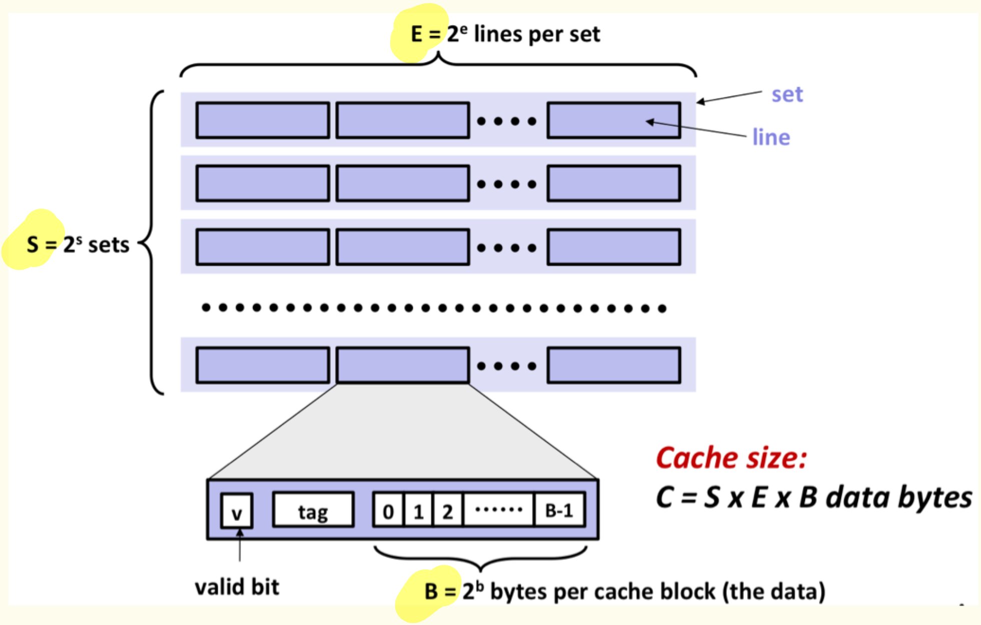

Cache
4
9
10
3
Smaller, faster, more expensive memory caches a subset of the blocks
10
Data is copied in block-sized transfer units
Memory
0 1 2 3
4 5 6 7
8 9 10 11
12 13 14 15
Larger, slower, cheaper memory viewed as partitioned into "blocks"





Block b is stored in cache

- Placement policy: determines where b goes
- Replacement policy: determines which block gets evicted (victim)

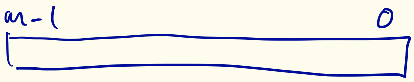


Cache size:

C = S x E x B data bytes

Define a cache organization by (S, E, B) + need $\underline{m}$: number of address bits

Address:

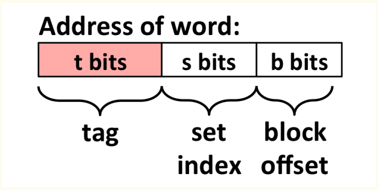


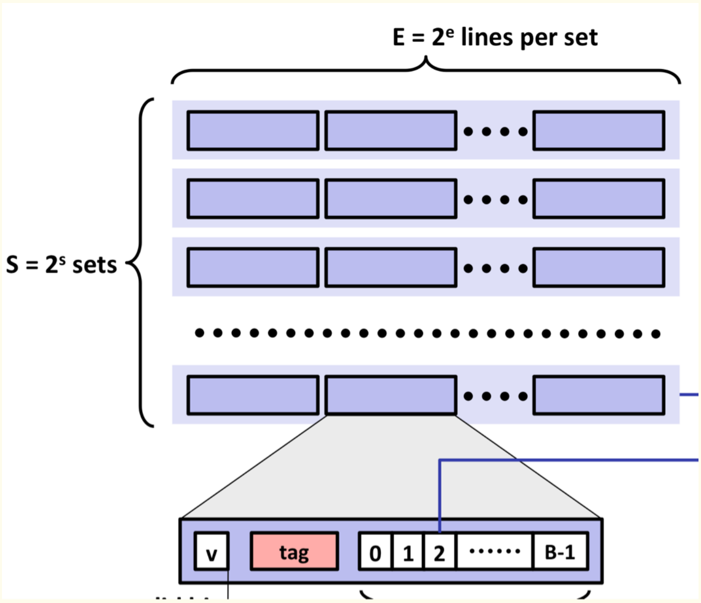
Derived from above:

$M = 2^m$: Maximum number of unique memory addresses

$s = \log_2(S)$: Number of bits to represent $\underline{\text{set}}$

$b = \log_2(B)$: Number of bits to represent block offset





① • *Locate set*

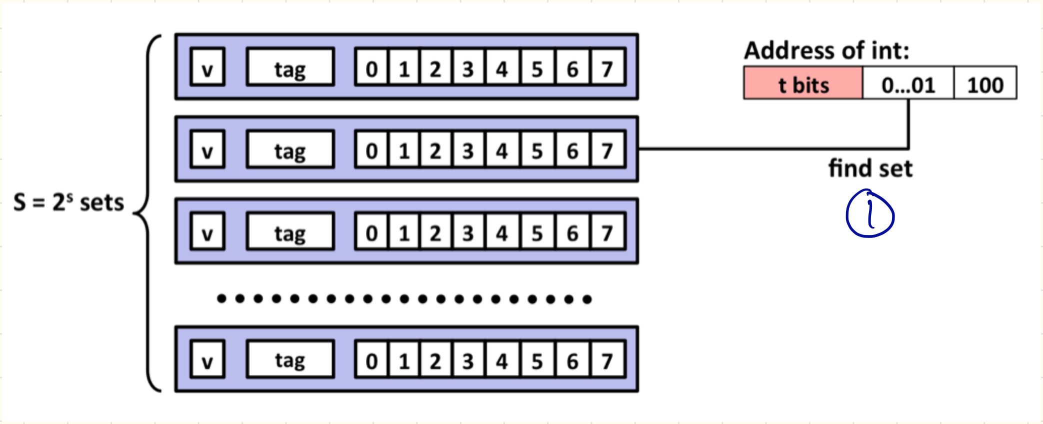
② • *Check if any line in set has matching tag*

③ • *Yes + line valid: hit*

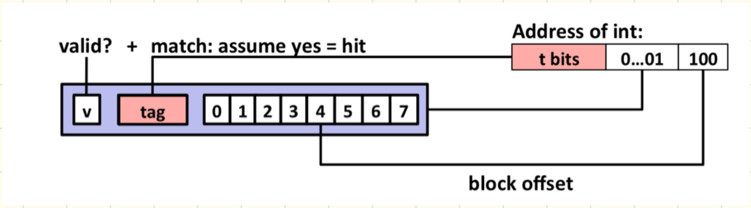
④ • *Locate data starting at offset*

Example 1:

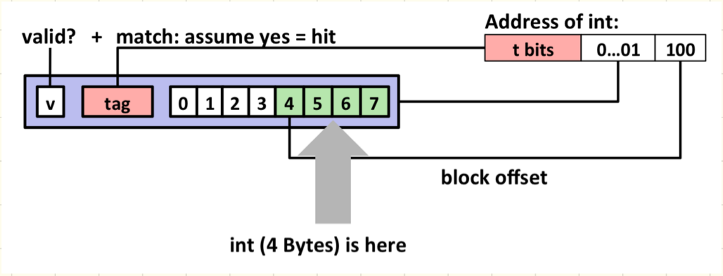
Direct mapped: One line per set
Assume: cache block size 8 bytes



②



③+④


valid? + match: assume yes = hit
Address of int:
t bits
0…01
100
v
tag
0 1 2 3 4 5 6 7
block offset
int (4 Bytes) is here